1. 概念

窗口分析函数：窗口函数也称为OLAP（OnlineAnalytical Processing）函数，是对一组值进行操作，不需要使用Group by子句对数据进行分组，还能在同一行返回原来行的列和使用聚合函数得到的聚合列。

官网：https://cwiki.apache.org/confluence/display/Hive/LanguageManual+WindowingAndAnalytics

2. sum, avg, max, min

数据准备：cookie1.txt

```
cookie1,2015-04-10,1
cookie1,2015-04-11,5
cookie1,2015-04-12,7
cookie1,2015-04-13,3
cookie1,2015-04-14,2
cookie1,2015-04-15,4
cookie1,2015-04-16,4
```

建表准备相关:

```
create database if not exists cookie_window_analytics;
use cookie_window_analytics;
drop table if exists cookie1;
create table cookie1(cookieid string, createtime string, pv int) row format
delimited fields terminated by ',';
load data local inpath "/home/bigdata/cookie1.txt" into table cookie1;
select * from cookie1;
```

2.1. sum

```
SELECT cookieid,createtime,pv,
SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime) AS pv1,  -- 默认为从起点到
当前行
SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS pv2,  --从起点到当前行，结果同pv1
SUM(pv) OVER(PARTITION BY cookieid) AS pv3, --分组内所有行
SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING
AND CURRENT ROW) AS pv4,    --当前行+往前3行
SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3 PRECEDING
AND 1 FOLLOWING) AS pv5,       --当前行+往前3行+往后1行
SUM(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN CURRENT ROW
AND UNBOUNDED FOLLOWING) AS pv6     --当前行+往后所有行
FROM cookie1 order by cookieid, createtime;
```

结果:

```
cookie1 2015-04-10      1       1       1       26      1       6       26
cookie1 2015-04-11      5       6       6       26      6       13      25
cookie1 2015-04-12      7       13      13      26      13      16      20
cookie1 2015-04-13      3       16      16      26      16      18      13
cookie1 2015-04-14      2       18      18      26      17      21      10
cookie1 2015-04-15      4       22      22      26      16      20      8
cookie1 2015-04-16      4       26      26      26      13      13      4
```

解释:

```
pv1：分组内从起点到当前行的pv累积，如，11号的pv1=10号的pv+11号的pv，12号=10号+11号+12号
pv2：同pv1
pv3：分组内(cookie1)所有的pv累加
pv4：分组内当前行+往前3行，11号=10号+11号， 12号=10号+11号+12号， 13号=10号+11号+12号+13号， 14号=11号+12号+13号+14号
pv5：分组内当前行+往前3行+往后1行，如，14号=11号+12号+13号+14号+15号=5+7+3+2+4=21
pv6：分组内当前行+往后所有行，如，13号=13号+14号+15号+16号=3+2+4+4=13，14号=14号+15号+16号=2+4+4=10
```

扩展：

```
如果不指定ROWS BETWEEN,默认为从起点到当前行;
如果不指定ORDER BY，则将分组内所有值累加;
关键是理解ROWS BETWEEN含义,也叫做WINDOW子句：
    PRECEDING：往前
    FOLLOWING：往后
    CURRENT ROW：当前行
    UNBOUNDED：起点，UNBOUNDED PRECEDING 表示从前面的起点， UNBOUNDED FOLLOWING：表示到后面的终点
```

2.2. avg，min，max，和 sum

其他AVG，MIN，MAX，和SUM用法一样。只需要把sum函数，改成avg，min，max，sum等就可以。

```
SELECT cookieid, createtime, pv,
round(AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime), 2) AS pv1,   --默认为从起点到当前行
round(AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN
UNBOUNDED PRECEDING AND CURRENT ROW), 2) AS pv2,    --从起点到当前行，结果同pv1
round(AVG(pv) OVER(PARTITION BY cookieid), 2) AS pv3,     --分组内所有行
round(AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3
PRECEDING AND CURRENT ROW), 2) AS pv4,   --当前行+往前3行
round(AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN 3
PRECEDING AND 1 FOLLOWING), 2) AS pv5,    --当前行+往前3行+往后1行
round(AVG(pv) OVER(PARTITION BY cookieid ORDER BY createtime ROWS BETWEEN
CURRENT ROW AND UNBOUNDED FOLLOWING), 2) AS pv6   --当前行+往后所有行
FROM cookie1;
```

执行结果：

```
cookie1 2015-04-16      4       3.71    3.71    3.71    3.25    3.25    4.0
cookie1 2015-04-15      4       3.67    3.67    3.71    4.0     4.0     4.0
cookie1 2015-04-14      2       3.6     3.6     3.71    4.25    4.2     3.33
cookie1 2015-04-13      3       4.0     4.0     3.71    4.0     3.6     3.25
cookie1 2015-04-12      7       4.33    4.33    3.71    4.33    4.0     4.0
cookie1 2015-04-11      5       3.0     3.0     3.71    3.0     4.33    4.17
cookie1 2015-04-10      1       1.0     1.0     3.71    1.0     3.0     3.71
```

3. row_number, rank, dense_rank

准备数据：cookie2.txt

```
cookie1,2015-04-10,1
cookie1,2015-04-11,5
cookie1,2015-04-12,7
cookie1,2015-04-13,3
cookie1,2015-04-14,2
cookie1,2015-04-15,4
cookie1,2015-04-16,4
cookie2,2015-04-10,2
cookie2,2015-04-11,3
cookie2,2015-04-12,5
cookie2,2015-04-13,6
cookie2,2015-04-14,3
cookie2,2015-04-15,9
cookie2,2015-04-16,7
```

建表导入数据相关操作：

```
create database if not exists cookie_window_analytics;
use cookie_window_analytics;
drop table if exists cookie2;
create table cookie2(cookieid string, createtime string, pv int) row format
delimited fields terminated by ',';
load data local inpath "/home/bigdata/cookie2.txt" into table cookie2;
select * from cookie2;
```

3.1. row_number

ROW_NUMBER() – 从1开始，按照顺序，生成分组内记录的序列 –比如，按照pv降序排列，生成分组内每天的pv名次，ROW_NUMBER() 的应用场景非常多，再比如，获取分组内排序第一的记录;获取一个session中的第一条refer等。

// 分组排序

```
SELECT cookieid, createtime, pv,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY pv desc) AS rn
FROM cookie2;
```

结果：

```
cookie1 2015-04-12      7       1
cookie1 2015-04-11      5       2
cookie1 2015-04-16      4       3
cookie1 2015-04-15      4       4
cookie1 2015-04-13      3       5
cookie1 2015-04-14      2       6
cookie1 2015-04-10      1       7
cookie2 2015-04-15      9       1
cookie2 2015-04-16      7       2
```

```
cookie2 2015-04-13      6       3
cookie2 2015-04-12      5       4
cookie2 2015-04-11      3       5
cookie2 2015-04-14      3       6
cookie2 2015-04-10      2       7
```

所以如果需要取每一组的前3名，只需要rn<=3即可

3.2. rank 和 dense_rank

RANK() 生成数据项在分组中的排名，排名相等会在名次中留下空位

DENSE_RANK() 生成数据项在分组中的排名，排名相等会在名次中不会留下空位

SQL语句实例：

```
SELECT cookieid, createtime, pv,
    RANK() OVER(PARTITION BY cookieid ORDER BY pv desc) AS rn1,
    DENSE_RANK() OVER(PARTITION BY cookieid ORDER BY pv desc) AS rn2,
    ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY pv DESC) AS rn3
FROM cookie2
WHERE cookieid = 'cookie1';
```

结果：

```
cookie1 2015-04-12      7       1       1       1
cookie1 2015-04-11      5       2       2       2
cookie1 2015-04-16      4       3       3       3
cookie1 2015-04-15      4       3       3       4
cookie1 2015-04-13      3       5       4       5
cookie1 2015-04-14      2       6       5       6
cookie1 2015-04-10      1       7       6       7
```

三者对比总结：

```
row_number   按顺序编号，不留空位
rank         按顺序编号，相同的值编相同号，留空位
dense_rank   按顺序编号，相同的值编相同的号，不留空位
```

4. ntile, cume_dist, percent_rank

数据准备：cookie3.txt

```
d1,user1,1000
d1,user2,2000
d1,user3,3000
d2,user4,4000
d2,user5,5000
```

建表导入数据相关操作：

```
create database if not exists cookie_window_analytics;
use cookie_window_analytics;
drop table if exists cookie3;
create table cookie3(dept string, userid string, sal int) row format delimited
fields terminated by ',';
load data local inpath "/home/bigdata/cookie3.txt" into table cookie3;
select * from cookie3;
```

4.1. ntile

```
NTILE(n)，用于将分组数据按照顺序切分成n片，返回当前切片值,如果切片不均匀，默认增加第一个切片的
分布
NTILE不支持ROWS BETWEEN，比如 NTILE(2) OVER(PARTITION BY cookieid ORDER BY
createtime ROWS BETWEEN 3 PRECEDING AND CURRENT ROW)
```

SQL语句实例：

```
SELECT cookieid,createtime,pv,
NTILE(2) OVER(PARTITION BY cookieid ORDER BY createtime) AS rn1,    --分组内将数据
分成2片
NTILE(3) OVER(PARTITION BY cookieid ORDER BY createtime) AS rn2,    --分组内将数据
分成3片
NTILE(4) OVER(ORDER BY createtime) AS rn3     --将所有数据分成4片
FROM cookie2
ORDER BY cookieid,createtime;
```

结果：

```
cookie1 2015-04-10      1       1       1       1
cookie1 2015-04-11      5       1       1       1
cookie1 2015-04-12      7       1       1       2
cookie1 2015-04-13      3       1       2       2
cookie1 2015-04-14      2       2       2       3
cookie1 2015-04-15      4       2       3       4
cookie1 2015-04-16      4       2       3       4
cookie2 2015-04-10      2       1       1       1
cookie2 2015-04-11      3       1       1       1
cookie2 2015-04-12      5       1       1       2
cookie2 2015-04-13      6       1       2       2
cookie2 2015-04-14      3       2       2       3
cookie2 2015-04-15      9       2       3       3
cookie2 2015-04-16      7       2       3       4
```

比如，统计一个 cookie，pv 数最多的前 1/3 的天

```
SELECT cookieid, createtime, pv,
NTILE(3) OVER(PARTITION BY cookieid ORDER BY pv DESC) AS rn
FROM cookie2;
```

结果：

```
cookie1 2015-04-12      7       1
cookie1 2015-04-11      5       1
cookie1 2015-04-16      4       1
cookie1 2015-04-15      4       2
cookie1 2015-04-13      3       2
cookie1 2015-04-14      2       3
cookie1 2015-04-10      1       3
cookie2 2015-04-15      9       1
cookie2 2015-04-16      7       1
cookie2 2015-04-13      6       1
cookie2 2015-04-12      5       2
cookie2 2015-04-11      3       2
cookie2 2015-04-14      3       3
cookie2 2015-04-10      2       3
```

其中：rn = 1 的记录，就是我们想要的结果

4.2. cume_dist

```
CUME_DIST 小于等于当前值的行数/分组内总行数
```

比如，统计小于等于当前薪水的人数，所占总人数的比例

```
SELECT  dept, userid, sal,
round(CUME_DIST() OVER(ORDER BY sal), 2) AS rn1,
round(CUME_DIST() OVER(PARTITION BY dept ORDER BY sal), 2) AS rn2
FROM cookie3;
```

结果：

```
d1      user1   1000    0.2     0.33
d1      user2   2000    0.4     0.67
d1      user3   3000    0.6     1.0
d2      user4   4000    0.8     0.5
d2      user5   5000    1.0     1.0
```

SQL语句实例：

```
SELECT  dept, userid, sal,
round(CUME_DIST() OVER(ORDER BY sal), 2) AS rn1,
round(CUME_DIST() OVER(PARTITION BY dept ORDER BY sal desc), 2)  AS rn2
FROM cookie3;
```

结果：

```
d1      user3   3000    0.6     0.33
d1      user2   2000    0.4     0.67
d1      user1   1000    0.2     1.0
d2      user5   5000    1.0     0.5
d2      user4   4000    0.8     1.0
```

4.3. percent_rank

PERCENT_RANK 分组内当前行的RANK值-1/分组内总行数-1

SQL语句实例：

```
SELECT  dept, userid, sal,
    PERCENT_RANK() OVER(ORDER BY sal) AS rn1,        --分组内
    RANK() OVER(ORDER BY sal) AS rn11,               --分组内RANK值
    SUM(1) OVER(PARTITION BY NULL) AS rn12,          --分组内总行数
    PERCENT_RANK() OVER(PARTITION BY dept ORDER BY sal) AS rn2
FROM cookie3;
```

结果：

```
d1      user1   1000    0.0     1       5       0.0
d1      user2   2000    0.25    2       5       0.5
d1      user3   3000    0.5     3       5       1.0
d2      user4   4000    0.75    4       5       0.0
d2      user5   5000    1.0     5       5       1.0
```

SQL语句实例：

```
SELECT  dept, userid, sal,
PERCENT_RANK() OVER(PARTITION BY dept ORDER BY sal) AS rn1,     --分组内
RANK() OVER(ORDER BY sal) AS rn11,                              --分组内RANK值
SUM(1) OVER(PARTITION BY NULL) AS rn12,                         --分组内总行数
PERCENT_RANK() OVER(PARTITION BY dept ORDER BY sal) AS rn2
FROM cookie3;
```

执行结果：

```
d2      user5   5000    1.0     5       5       1.0
d2      user4   4000    0.0     4       5       0.0
d1      user3   3000    1.0     3       5       1.0
d1      user2   2000    0.5     2       5       0.5
d1      user1   1000    0.0     1       5       0.0
```

5. lag, lead, frist_value, last_value

数据准备：cookie4.txt

```
cookie1,2015-04-10 10:00:02,url2
cookie1,2015-04-10 10:00:00,url1
cookie1,2015-04-10 10:03:04,url3
cookie1,2015-04-10 10:50:05,url6
cookie1,2015-04-10 11:00:00,url7
cookie1,2015-04-10 10:10:00,url4
cookie1,2015-04-10 10:50:01,url5
cookie2,2015-04-10 10:00:02,url22
cookie2,2015-04-10 10:00:00,url11
```

```
cookie2,2015-04-10 10:03:04,url33
cookie2,2015-04-10 10:50:05,url66
cookie2,2015-04-10 11:00:00,url77
cookie2,2015-04-10 10:10:00,url44
cookie2,2015-04-10 10:50:01,url55
```

建表导入数据相关操作：

```
create database if not exists cookie_window_analytics;
use cookie_window_analytics;
drop table if exists cookie4;
create table cookie4(cookieid string, createtime string, url string) row format
delimited fields terminated by ',';
load data local inpath "/home/bigdata/cookie4.txt" into table cookie4;
select * from cookie4;
```

5.1. lag

LAG(col,n,DEFAULT) 用于统计窗口内往上第n行值
第一个参数为列名，
第二个参数为往上第n行（可选，默认为1），
第三个参数为默认值（当往上第n行为NULL时候，取默认值，如不指定，则为NULL）

SQL语句实例：

```
SELECT cookieid, createtime, url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
LAG(createtime,1,'1970-01-01 00:00:00') OVER(PARTITION BY cookieid ORDER BY
createtime) AS last_1_time,
LAG(createtime,2) OVER(PARTITION BY cookieid ORDER BY createtime) AS last_2_time
FROM cookie4;
```

结果数据：

```
cookieid createtime             url     rn      last_1_time
last_2_time
cookie1 2015-04-10 10:00:00     url1    1       1970-01-01 00:00:00     NULL
cookie1 2015-04-10 10:00:02     url2    2       2015-04-10 10:00:00     NULL
cookie1 2015-04-10 10:03:04     url3    3       2015-04-10 10:00:02     2015-04-
10 10:00:00
cookie1 2015-04-10 10:10:00     url4    4       2015-04-10 10:03:04     2015-04-
10 10:00:02
cookie1 2015-04-10 10:50:01     url5    5       2015-04-10 10:10:00     2015-04-
10 10:03:04
cookie1 2015-04-10 10:50:05     url6    6       2015-04-10 10:50:01     2015-04-
10 10:10:00
cookie1 2015-04-10 11:00:00     url7    7       2015-04-10 10:50:05     2015-04-
10 10:50:01
cookie2 2015-04-10 10:00:00     url11   1       1970-01-01 00:00:00     NULL
cookie2 2015-04-10 10:00:02     url22   2       2015-04-10 10:00:00     NULL
cookie2 2015-04-10 10:03:04     url33   3       2015-04-10 10:00:02     2015-04-
10 10:00:00
```

```
cookie2 2015-04-10 10:10:00     url44   4       2015-04-10 10:03:04     2015-04-
10 10:00:02
cookie2 2015-04-10 10:50:01     url55   5       2015-04-10 10:10:00     2015-04-
10 10:03:04
cookie2 2015-04-10 10:50:05     url66   6       2015-04-10 10:50:01     2015-04-
10 10:10:00
cookie2 2015-04-10 11:00:00     url77   7       2015-04-10 10:50:05     2015-04-
10 10:50:01
```

解释:

```
last_1_time: 指定了往上第1行的值，default为'1970-01-01 00:00:00'
             cookie1第一行，往上1行为NULL,因此取默认值 1970-01-01 00:00:00
             cookie1第三行，往上1行值为第二行值，2015-04-10 10:00:02
             cookie1第六行，往上1行值为第五行值，2015-04-10 10:50:01
last_2_time: 指定了往上第2行的值，为指定默认值
             cookie1第一行，往上2行为NULL
             cookie1第二行，往上2行为NULL
             cookie1第四行，往上2行为第二行值，2015-04-10 10:00:02
             cookie1第七行，往上2行为第五行值，2015-04-10 10:50:01
```

5.2. lead

与LAG相反
LEAD(col,n,DEFAULT) 用于统计窗口内往下第n行值
第一个参数为列名，
第二个参数为往下第n行（可选，默认为1），
第三个参数为默认值（当往下第n行为NULL时候，取默认值，如不指定，则为NULL)

SQL语句实例:

```
SELECT cookieid, createtime, url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
LEAD(createtime,1,'1970-01-01 00:00:00') OVER(PARTITION BY cookieid ORDER BY
createtime) AS next_1_time,
LEAD(createtime,2) OVER(PARTITION BY cookieid ORDER BY createtime) AS
next_2_time
FROM cookie4;
```

结果:

```
cookieid createtime             url     rn      next_1_time
next_2_time
cookie1 2015-04-10 10:00:00     url1    1       2015-04-10 10:00:02     2015-04-
10 10:03:04
cookie1 2015-04-10 10:00:02     url2    2       2015-04-10 10:03:04     2015-04-
10 10:10:00
cookie1 2015-04-10 10:03:04     url3    3       2015-04-10 10:10:00     2015-04-
10 10:50:01
cookie1 2015-04-10 10:10:00     url4    4       2015-04-10 10:50:01     2015-04-
10 10:50:05
cookie1 2015-04-10 10:50:01     url5    5       2015-04-10 10:50:05     2015-04-
10 11:00:00
```

```
cookie1 2015-04-10 10:50:05     url6    6       2015-04-10 11:00:00     NULL
cookie1 2015-04-10 11:00:00     url7    7       1970-01-01 00:00:00     NULL
cookie2 2015-04-10 10:00:00     url11   1       2015-04-10 10:00:02     2015-04-
10 10:03:04
cookie2 2015-04-10 10:00:02     url22   2       2015-04-10 10:03:04     2015-04-
10 10:10:00
cookie2 2015-04-10 10:03:04     url33   3       2015-04-10 10:10:00     2015-04-
10 10:50:01
cookie2 2015-04-10 10:10:00     url44   4       2015-04-10 10:50:01     2015-04-
10 10:50:05
cookie2 2015-04-10 10:50:01     url55   5       2015-04-10 10:50:05     2015-04-
10 11:00:00
cookie2 2015-04-10 10:50:05     url66   6       2015-04-10 11:00:00     NULL
cookie2 2015-04-10 11:00:00     url77   7       1970-01-01 00:00:00     NULL
```

5.3. first_value

first_value：取分组内排序后，截止到当前行，第一个值

SQL语句实例：

```
SELECT cookieid, createtime, url,
ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
FIRST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS first1
FROM cookie4;
```

结果：

```
cookie1 2015-04-10 10:00:00     url1    1       url1
cookie1 2015-04-10 10:00:02     url2    2       url1
cookie1 2015-04-10 10:03:04     url3    3       url1
cookie1 2015-04-10 10:10:00     url4    4       url1
cookie1 2015-04-10 10:50:01     url5    5       url1
cookie1 2015-04-10 10:50:05     url6    6       url1
cookie1 2015-04-10 11:00:00     url7    7       url1
cookie2 2015-04-10 10:00:00     url11   1       url11
cookie2 2015-04-10 10:00:02     url22   2       url11
cookie2 2015-04-10 10:03:04     url33   3       url11
cookie2 2015-04-10 10:10:00     url44   4       url11
cookie2 2015-04-10 10:50:01     url55   5       url11
cookie2 2015-04-10 10:50:05     url66   6       url11
cookie2 2015-04-10 11:00:00     url77   7       url11
```

5.4. last_value

last_value：取分组内排序后，截止到当前行，最后一个值

SQL语句实例：

```
SELECT cookieid, createtime, url,
    ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY  ) AS rn,
    LAST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS last1
FROM cookie4;
```

结果数据：

```
cookie1 2015-04-10 10:00:00     url1    1       url1
cookie1 2015-04-10 10:00:02     url2    2       url2
cookie1 2015-04-10 10:03:04     url3    3       url3
cookie1 2015-04-10 10:10:00     url4    4       url4
cookie1 2015-04-10 10:50:01     url5    5       url5
cookie1 2015-04-10 10:50:05     url6    6       url6
cookie1 2015-04-10 11:00:00     url7    7       url7
cookie2 2015-04-10 10:00:00     url11   1       url11
cookie2 2015-04-10 10:00:02     url22   2       url22
cookie2 2015-04-10 10:03:04     url33   3       url33
cookie2 2015-04-10 10:10:00     url44   4       url44
cookie2 2015-04-10 10:50:01     url55   5       url55
cookie2 2015-04-10 10:50:05     url66   6       url66
cookie2 2015-04-10 11:00:00     url77   7       url77
```

问题：如果不指定ORDER BY，则默认按照记录在文件中的偏移量进行排序，会出现错误的结果

// 求得每组的最后一个值： 排倒序，然后取 FIRST_VALUE

SQL语句：

```
SELECT cookieid, createtime, url,
    ROW_NUMBER() OVER(PARTITION BY cookieid ORDER BY createtime) AS rn,
    LAST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime) AS last1,
    FIRST_VALUE(url) OVER(PARTITION BY cookieid ORDER BY createtime DESC) AS
last2
FROM cookie4
ORDER BY cookieid,createtime;
```

结果：

```
cookie1 2015-04-10 10:00:00     url1    1       url1    url7
cookie1 2015-04-10 10:00:02     url2    2       url2    url7
cookie1 2015-04-10 10:03:04     url3    3       url3    url7
cookie1 2015-04-10 10:10:00     url4    4       url4    url7
cookie1 2015-04-10 10:50:01     url5    5       url5    url7
cookie1 2015-04-10 10:50:05     url6    6       url6    url7
cookie1 2015-04-10 11:00:00     url7    7       url7    url7
cookie2 2015-04-10 10:00:00     url11   1       url11   url77
cookie2 2015-04-10 10:00:02     url22   2       url22   url77
cookie2 2015-04-10 10:03:04     url33   3       url33   url77
cookie2 2015-04-10 10:10:00     url44   4       url44   url77
cookie2 2015-04-10 10:50:01     url55   5       url55   url77
cookie2 2015-04-10 10:50:05     url66   6       url66   url77
cookie2 2015-04-10 11:00:00     url77   7       url77   url77
```

6. grouping sets, grouping__id, cube, rollup

这几个分析函数通常用于OLAP中，不能累加，而且需要根据不同维度上钻和下钻的指标统计，比如，分小时、天、月的UV数

官网介绍：https://cwiki.apache.org/confluence/display/Hive/Enhanced+Aggregation%2C+Cube%2C+Grouping+and+Rollup

数据准备：cookie5.txt

```
2015-03,2015-03-10,cookie1
2015-03,2015-03-10,cookie5
2015-03,2015-03-12,cookie7
2015-04,2015-04-12,cookie3
2015-04,2015-04-13,cookie2
2015-04,2015-04-13,cookie4
2015-04,2015-04-16,cookie4
2015-03,2015-03-10,cookie2
2015-03,2015-03-10,cookie3
2015-04,2015-04-12,cookie5
2015-04,2015-04-13,cookie6
2015-04,2015-04-15,cookie3
2015-04,2015-04-15,cookie2
2015-04,2015-04-16,cookie1
```

建表导入数据相关：

```
create database if not exists cookie_window_analytics;
use cookie_window_analytics;
drop table if exists cookie5;
create table cookie5(month string, day string, cookieid string) row format
delimited fields terminated by ',';
load data local inpath "/home/bigdata/cookie5.txt" into table cookie5;
select * from cookie5;
```

6.1. grouping sets

在一个GROUP BY查询中，根据不同的维度组合进行聚合，等价于将不同维度的GROUP BY结果集进行UNION ALL

SQL语句实例：

```
SELECT  month, day,
COUNT(DISTINCT cookieid) AS uv,
GROUPING__ID
FROM cookie5
GROUP BY month,day
GROUPING SETS (month,day)
ORDER BY GROUPING__ID;
```

其中的 GROUPING__ID，表示结果属于哪一个分组集合。

结果：

```
2015-04 NULL            6       1
2015-03 NULL            5       1
NULL    2015-04-16      2       2
NULL    2015-04-15      2       2
NULL    2015-04-13      3       2
NULL    2015-04-12      2       2
NULL    2015-03-12      1       2
NULL    2015-03-10      4       2
```

其实这个SQL语句等价于下面这个SQL：

```
SELECT month,NULL,COUNT(DISTINCT cookieid) AS uv,1 AS GROUPING__ID FROM cookie5
GROUP BY month
UNION ALL
SELECT NULL,day,COUNT(DISTINCT cookieid) AS uv,2 AS GROUPING__ID FROM cookie5
GROUP BY day;
```

执行结果：

```
NULL    2015-03-10      4       2
NULL    2015-03-12      1       2
NULL    2015-04-12      2       2
NULL    2015-04-13      3       2
NULL    2015-04-15      2       2
NULL    2015-04-16      2       2
2015-03 NULL            5       1
2015-04 NULL            6       1
```

SQL语句：

```
SELECT  month, day,
COUNT(DISTINCT cookieid) AS uv,
GROUPING__ID
FROM cookie5
GROUP BY month,day
GROUPING SETS (month,day,(month,day))
ORDER BY GROUPING__ID;
```

其中的 GROUPING__ID，表示结果属于哪一个分组集合。

结果数据：

```
2015-03 2015-03-10      4       0
2015-04 2015-04-16      2       0
2015-04 2015-04-13      3       0
2015-04 2015-04-12      2       0
2015-04 2015-04-15      2       0
2015-03 2015-03-12      1       0
2015-03 NULL            5       1
2015-04 NULL            6       1
NULL    2015-04-16      2       2
```

```
NULL    2015-04-15      2       2
NULL    2015-04-13      3       2
NULL    2015-04-12      2       2
NULL    2015-03-12      1       2
NULL    2015-03-10      4       2
```

等价于：

```
SELECT month,NULL,COUNT(DISTINCT cookieid) AS uv,1 AS GROUPING__ID FROM cookie5
GROUP BY month
UNION ALL
SELECT NULL,day,COUNT(DISTINCT cookieid) AS uv,2 AS GROUPING__ID FROM cookie5
GROUP BY day
UNION ALL
SELECT month,day,COUNT(DISTINCT cookieid) AS uv,3 AS GROUPING__ID FROM cookie5
GROUP BY month,day;
```

6.2. cube

根据GROUP BY的维度的所有组合进行聚合

SQL语句：

```
SELECT  month, day,
COUNT(DISTINCT cookieid) AS uv,
GROUPING__ID
FROM cookie5
GROUP BY month,day
WITH CUBE
ORDER BY GROUPING__ID;
```

结果：

```
2015-03 2015-03-10      4       0
2015-04 2015-04-16      2       0
2015-04 2015-04-13      3       0
2015-04 2015-04-12      2       0
2015-04 2015-04-15      2       0
2015-03 2015-03-12      1       0
2015-03 NULL            5       1
2015-04 NULL            6       1
NULL    2015-04-16      2       2
NULL    2015-04-15      2       2
NULL    2015-04-13      3       2
NULL    2015-04-12      2       2
NULL    2015-03-12      1       2
NULL    2015-03-10      4       2
NULL    NULL            7       3
```

等价于：

```
SELECT NULL,NULL,COUNT(DISTINCT cookieid) AS uv,0 AS GROUPING__ID FROM cookie5
UNION ALL
SELECT month,NULL,COUNT(DISTINCT cookieid) AS uv,1 AS GROUPING__ID FROM cookie5
GROUP BY month
UNION ALL
SELECT NULL,day,COUNT(DISTINCT cookieid) AS uv,2 AS GROUPING__ID FROM cookie5
GROUP BY day
UNION ALL
SELECT month,day,COUNT(DISTINCT cookieid) AS uv,3 AS GROUPING__ID FROM cookie5
GROUP BY month,day;
```

6.3. rollup

是CUBE的子集，以最左侧的维度为主，从该维度进行层级聚合

比如，以month维度进行层级聚合，SQL语句：

```
SELECT  month, day, COUNT(DISTINCT cookieid) AS uv, GROUPING__ID
FROM cookie5
GROUP BY month,day WITH ROLLUP  ORDER BY GROUPING__ID;
```

可以实现这样的上钻过程：月天的UV->月的UV->总UV

结果：

```
2015-04 2015-04-16      2       0
2015-04 2015-04-15      2       0
2015-04 2015-04-13      3       0
2015-04 2015-04-12      2       0
2015-03 2015-03-12      1       0
2015-03 2015-03-10      4       0
2015-04 NULL            6       1
2015-03 NULL            5       1
NULL    NULL            7       3
```

把month和day调换顺序，则以day维度进行层级聚合：SQL语句：

```
SELECT  day, month, COUNT(DISTINCT cookieid) AS uv, GROUPING__ID
FROM cookie5
GROUP BY day,month  WITH ROLLUP  ORDER BY GROUPING__ID;
```

可以实现这样的上钻过程：天月的UV->天的UV->总UV
这里，根据天和月进行聚合，和根据天聚合结果一样，因为有父子关系，如果是其他维度组合的话，就会不一样